Tema 3 LFA

REGEX Parser

Calmis Alina 335CB

Pentru implementarea temei REGEX-> NFA-> DFA am folosit pe post de schelet de cod laboratorul 12.

Clasele create : Stack, Expr, Or, And, Par, Alpha, Kleene, Parser si NFA.

1. Clasa **Stack**:
   Implementeaza o stiva, cu toate metodele aferente ei: init, peek, push, pop, empty si search

2. Clasa **Expr**:
   Instantiaza tipul expresie

3. Clasele **Or, And, Par, Alpha, Kleene**:
   Toate extind clasa Expr si instantiaza corespunzator fiecare tip de expresie

4. Clasa **Parser**:
   - *Init* -> initiaza toate variabilele clasei
   - *nextState* -> pentu fiecare tranzitie din dictionarul de tranzitii, verfica daca se analizeaza starea curenta, daca se analizeaza caracterul necesar, adica ultimul, atunci tranzitia este pusa pe stiva
   - *parReduction, alphaReduction, orReduction, br_orReduction, andReduction, kleeneReduction* -> toate extrag de pe stiva expresiile si simbolurile necesare, inlocuindu-le cu regExp aferenta
   - *reduce* -> reduce expresia de pe stiva, apeland una din metodele de mai sus:
   - *parse* -> parcurge expresia data ca input , de la inceput pentru starea initiala, apoi, atat timp cate expresia nu este goala, se extrage starea urmatoare, daca nu exista o astfel de stare, sau stiva s-a eliberat inainte de parcurgerea intregii expresii, atunci se intrerupe cautarea, in alt caz, expresia se redeuce, actualizand constant stiva

5. Clasa **NFA**:
   - *Init* -> initiaza toate variabilele clasei
   - *checkOperation* -> verifica daca este instanta a Par, And, Or, Kleene si returneaza un vector cu caracterul specific si expresiile aferente
   - *readExpr* -> analizeaza expresia regulata primita ca input, verifica tipul expresiei, daca nu este None (nu e litera), atunci selecteaza Caracterul specific operatiei, si analizeaza in continuare expresia, pana se gasese o litera. Cand a fost identificata litera, se creste numarul de stari, se creeaza un nfa pentru litera curenta , format din starea initiala, tranzitia si starea finala. Se adauga pe stiva si se intoarce la pasul anterior, unde analizeaza operatia/ caracterul specific in functie de cele acceptate , adica kleene, and sau or. Pentru fiecare din aceste cazuri se extrag nfa-urile anterior create necesare de pe stiva, se actualizeaza toate datele si se creeaza un nou nfa care este in continuare propagat pana se termina de analizat expresia.

- *checkDoubledTransitions* -> modifica vectorul de tranziti, astfel incat daca pentru aceeasi stare initiala si aceeasi tranzitie sunt doua sau mai multe stari urmatoare, atunci toate starile urmatoare se adauga la prima lista identificata
- *copmuteTransitions* -> creeaza vectorul final de tranzitii

6. Metoda *WriteOutput* -> scrie outputul NFA-ului final in fisierul dat in argumentul 2, deoarece la testul 10 apare o eroare, am inclus aici si scrierea in fisierul 2 a celor 3 componente necesare, altfel imi pica checker-ul

7. Metoda *read_input* -> citeste inputul, creaaza alfabetul , apeleaza functia de creare a regExp, de creare a nfa, si scriere output.

Pentru conversia NFA – DFA , am importat codul de la tema2 , putin modificat , care contine doua clase, DFA si NFA.

1. Clasa DFA :
   Contine toate elemnetele necesare pentru definirea unui automat determinist finit : numarul de stari, starile finale, codificarea starilor si delta(tranzitiile)
2. Clasa NFA:
   - *Init* -> initiaza un NFA
   - *epsilon* -> verifica recursiv inchiderile epsilon a starii curente, primite ca input
   - *epsilonClosure* -> calculeaza inchiderile pentru fiecare stare a automatului
   - *getDFA* -> creeaza DFA – ul, incepand cu starea 0. Pentru fiecare litera din alfabet, pentru fiecare stare din inchidere verifica daca exista tranzitia in delta si creeaza o noua stare. Pornind de la aceasta se verifica inchiderile epsilon a fiecarei stari in parte si include tranzitiile in dictionarul final, care reprezinta o noua delta a DFA - ului . Verifica tranzitiile neincluse in delta, creeaza un sink state in care le cumuleaza pe toate si face update la dictionarul de tranzitii.
   - *createDFA* -> citeste NFA-ul din fisier, calculeaza inchiderile epsilon, creeaza DFA-ul si scrie DFA-ul nou creat in fisier